How I Built A Document Classification System using Deep Convolutional Neural Networks!

sambal shikhar · Follow

Published in Analytics Vidhya · 9 min read · Nov 1, 2019

With the advent of Deep Learning ,OCR's(Optical Character Recognition) Tools have improved a lot and become more efficient.

Did you guys know that there is an inbuilt OCR feature in Google Docs itself?

So how does a Document Classification Pipe Line help in improving OCR's ?

In content detection OCR's we can use the Document Classification pipeline for indexing different documents such that the OCR can operate efficiently for different layouts separately.Building a general OCR's for all kinds of documents such as "Research Papers,News paper articles,Reports,Forms,Resume" is a very tough task ,but if we can divide the algorithms it can do all the task efficiently.

Document classification prior to OCR's functioning can help us preprocess different kinds of document images in different ways such that the task of OCR becomes easier.

Lets get started!

Data Set Description

The RVL-CDIP (Ryerson Vision Lab Complex Document Information Processing) dataset consists of 400,000 grayscale images in 16 classes, with 25,000 images per class. There are 320,000 training images, 40,000 validation images, and 40,000 test images. The images are sized so their largest dimension does not exceed 1000 pixels.

Lets use Deep Convolutional Neural Networks to build our Document Classification System with an accuracy of over 90% using only 1/3 of the data!

The 16 classes are as follows :

1. letter
2. form
3. email
4. handwritten
5. advertisement
6. scientific report
7. scientific publication
8. specification
9. file folder
10. news article
11. budget
12. invoice

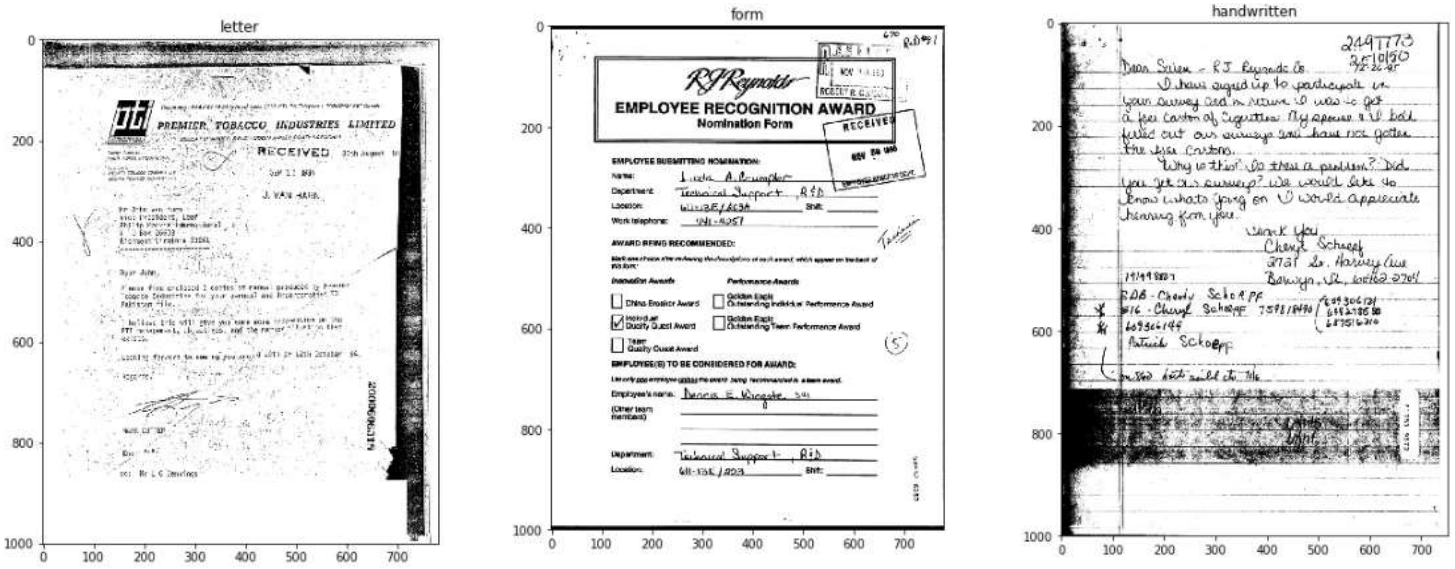
13. presentation

14. questionnaire

15. resume

16. memo

Lets Visualize some of the samples from the data set:



Creating Structured Data

The label files list the images and their categories in the following format:

```
path/to/the/image.tif category
```

We create a script such that we get two separate pickel files for labels as well as paths to images

```
import os
import joblib
count=0
labels=[]
```

```
directories=
['C:/Users/sambal/Desktop/labels/train.txt','C:/Users/sambal/Desktop/
labels/test.txt','C:/Users/sambal/Desktop/labels/val.txt']
train_path=[]
test_path=[]
val_path=[]
paths=[train_path,test_path,val_path]
y_train=[]
y_test=[]
y_cv=[]
labels=[y_train,y_test,y_cv]

for i in range(3):
        label=[]
        with open(directories[i],'r') as f:
            for line in f:
                label.append(line)

path=[]

for x in label:

category=x.split(" ")[1][-2::-1][::-1]
            labels[i].append(category)
            y=x.split(" ")[0]
            paths[i].append(y)

for i in range(3):
    print(paths[i][0],labels[i][0])

joblib.dump(labels,"labels")
joblib.dump(paths,"paths")
```

Once we have our labels and paths separated into files we can now move forward and filter our data into three parts-Train,Test and Cv.

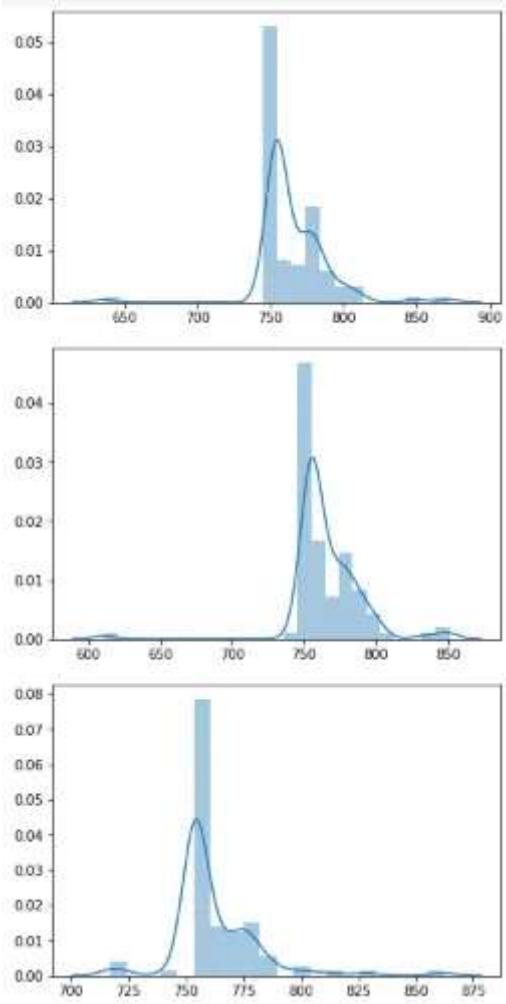
Lets check if there is any variation in image sizes using some samples of image

```
train_width=[]

test_width=[]

cv_width=[]

count=0

for image in train[:100]:
```

```
im = cv2.imread(image,cv2.IMREAD_GRAYSCALE)

train_width.append(im.shape[1])

for image in test[:100]:

im = cv2.imread(image,cv2.IMREAD_GRAYSCALE)

test_width.append(im.shape[1])

count+=1

for image in cv[:100]:

im = cv2.imread(image,cv2.IMREAD_GRAYSCALE)

cv_width.append(im.shape[1])
```

Lets visualize the image width distributions.

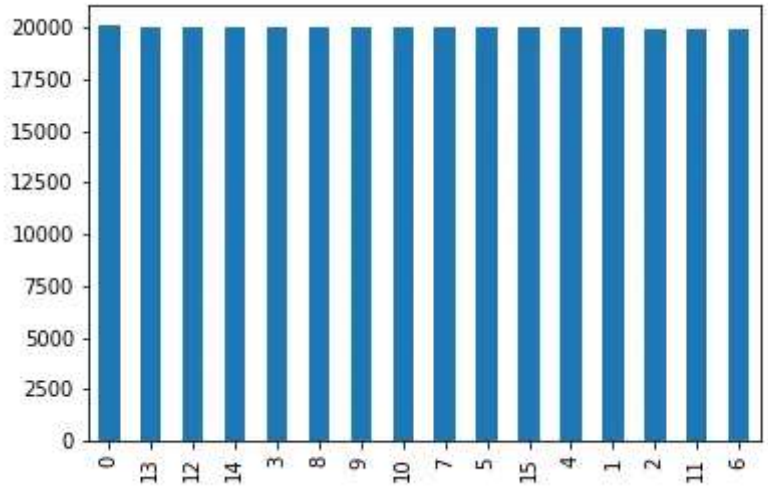
```
import seaborn as sb
from matplotlib import pyplot as plt

width=(train_width,test_width,cv_width)

for x in width:
   sb.distplot(x,kde = True)
   plt.show()
```



We see that there is a lot of variance in the width of the images,for all of the three Test,Train and CV data images.

So resizing images will be an important part of this project.

Checking for class imbalances



This is a fairly balanced data set with all the classes having equal representation.

Building Generators and File structures

We are going to have 3 folders, namely train,test and cv.

Each folder will have a 16 files,representing each class starting from 0 to 15

We move all the data with respect to there labels and into there respective folders.

Now we are moving only 10k data per Class for training into these folders due to computational constraints !

As the data is huge we would need a Keras generator to load the data batches from the disk to the model for efficient I/O.So we initialize Keras data generators by also including data normalization aspect such that our data is in range of 0 to 1,dividing by 255 .

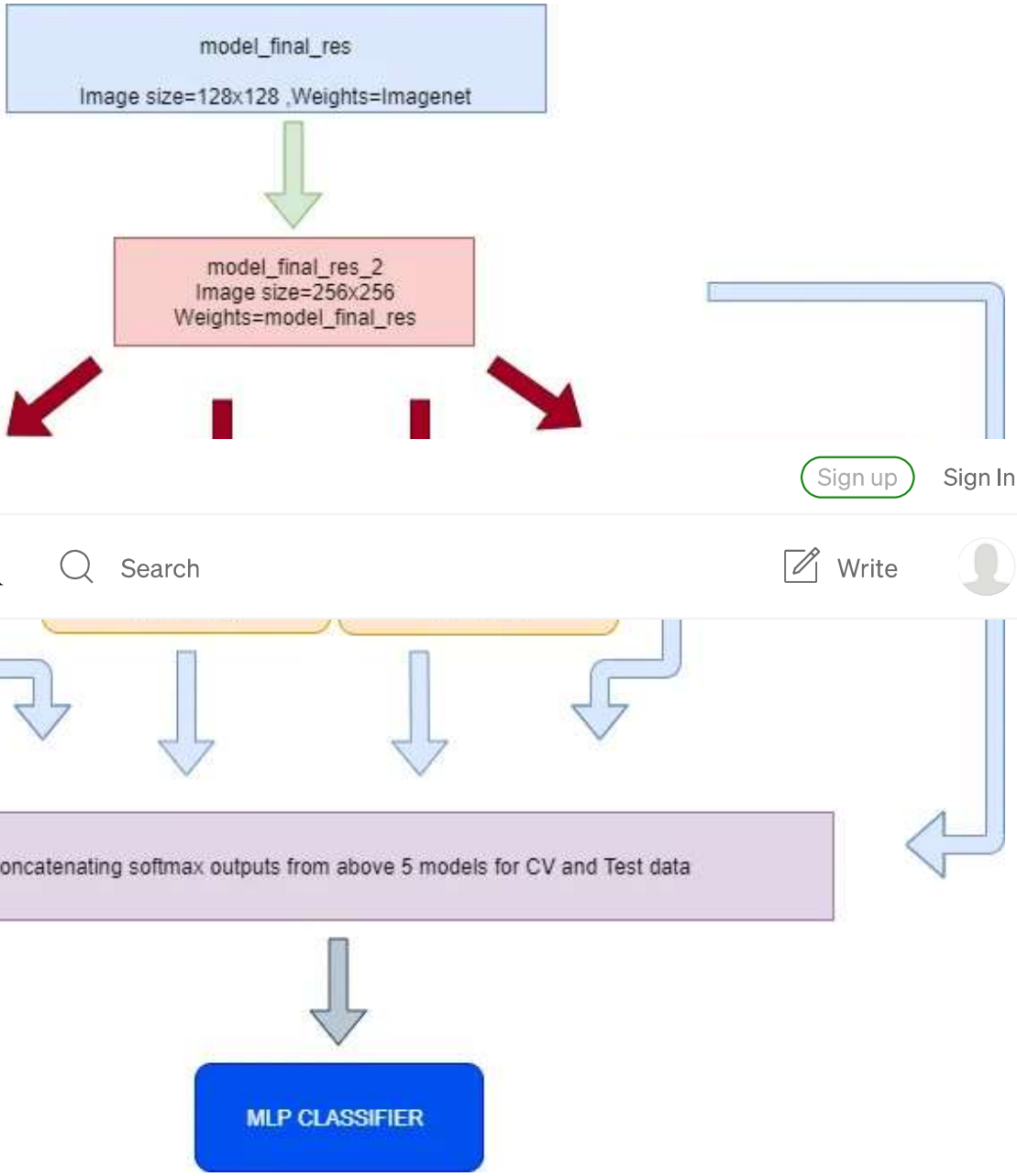
```
from keras.preprocessing.image import ImageDataGenerator
```

```
train_datagen = ImageDataGenerator(
rescale = 1./255)
test_datagen = ImageDataGenerator(
rescale = 1./255)
cv_datagen = ImageDataGenerator(
rescale = 1./255)
```

```
train_it = train_datagen.flow_from_directory('dataset/train/',batch_size=16,target_size=(128,128))
# load and iterate validation dataset
test_it = test_datagen.flow_from_directory('dataset/test/',batch_size=16,target_size=(128,128))
# load and iterate test dataset
cv_it = cv_datagen.flow_from_directory('dataset/cv/', batch_size=16,target_size=(128,128))
```

```
Found 159951 images belonging to 16 classes.
Found 39997 images belonging to 16 classes.
Found 39995 images belonging to 16 classes.
```

The Architecture



Oh ! well okay this thing that i dumped above seems super complicated right ?

So lets break it down piece by piece !

BASE MODELS

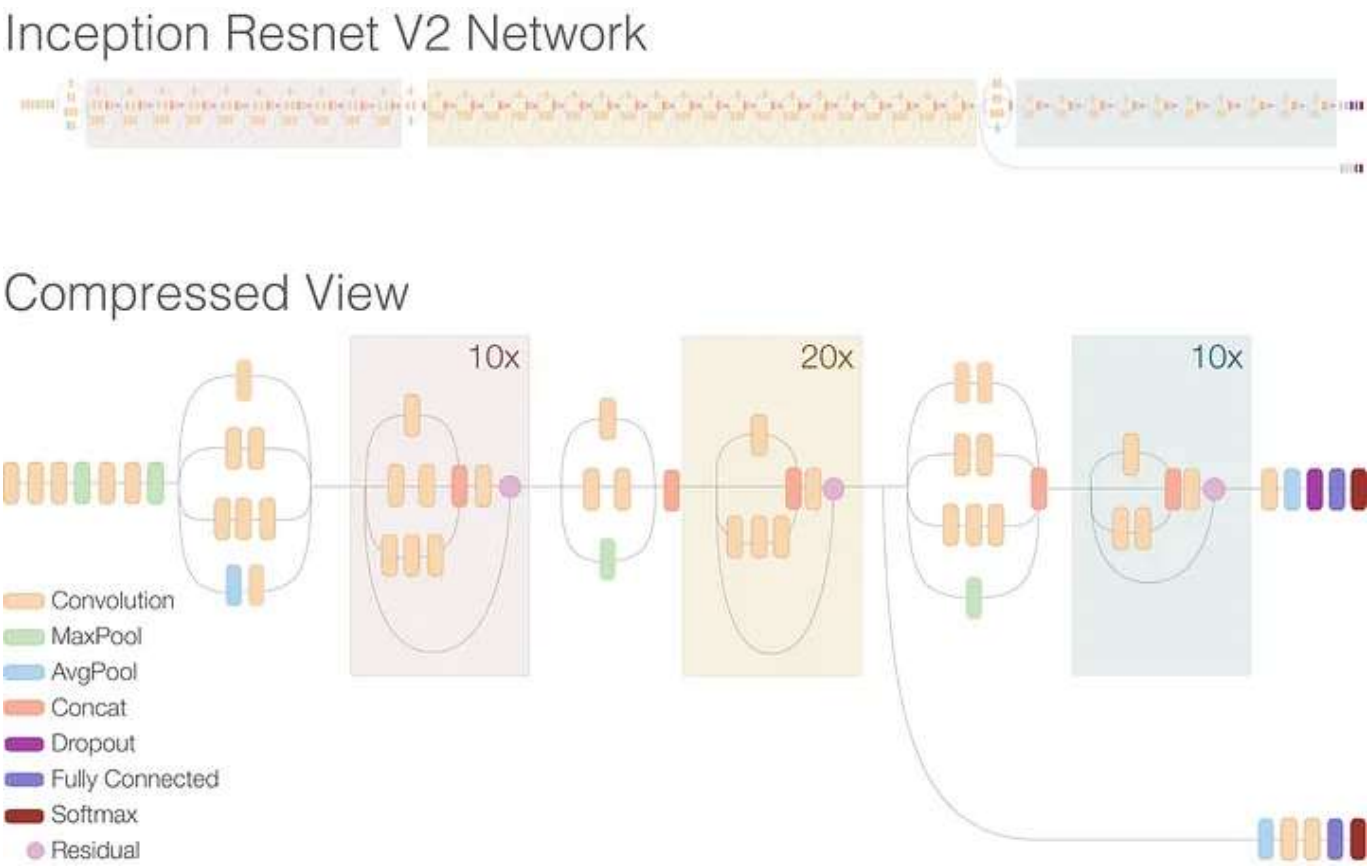
1. model_final_res

We are using a **Inceptionresnet-V2** initialized with weights trained on imgenet.We will resize our image into size of 128x128 and train this model.We unfreeze all the layers and train.

You might have heard of Resnets and Inception-net but what is a inception_resnet_v2?

As the name suggests it basically a Inception-net combined with residual modules concept from the Resnet.

Residual connections allow shortcuts in the model and have allowed researchers to successfully train even deeper neural networks, which have lead to even better performance. This has also enabled significant simplification of the Inception blocks.



```
model_res =keras.applications.inception_resnet_v2.InceptionResNetV2(weights = 'imagenet',
                                                                     include_top=False, input_shape = (None,None, 3))
```

We use the Keras application API to import the inception_resnet_v2 model with imagenet weights and also set it with variable input shape as using (None,None,3).We will be doing something interesting later on this blog.

```
for layer in model_res.layers:
    layer.trainable=True
#Adding custom Layers
x = model_res.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(128, activation="relu")(x)
x=Dropout(0.4)(x)
predictions = Dense(16, activation="softmax")(x)

# creating the final model
model_final_res = Model(input = model_res.input, output = predictions)

# compile the model
model_final_res.compile(loss = "categorical_crossentropy", optimizer=SGD(lr=0.1,momentum=0.0,nesterov=False)
, metrics=["accuracy"])
```

We remove the top of the inception_resnet_v2 model and attach 2 dense layers with Dropout and finally attach a Softmax layer of size 16.

We will be using the old classic SGD with
lr=0.1,momentum=0.0,nesterov=False

```
from keras.callbacks import ReduceLROnPlateau

reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.1,
                              patience=2, min_lr=0.00000001)
mcp_save = ModelCheckpoint('model_res.hdf5', save_best_only=True, monitor='val_accuracy', mode='max')
```

```
history=model_final_res.fit_generator(
train_it,
steps_per_epoch =159951/128,
epochs=30,
validation_data=cv_it,
validation_steps=39995/128,
callbacks=[mcp_save,reduce_lr]
)
```

We will penalize our learning rate,if the validation accuracy does not improve after 2 epoch by 10% and finally train it for 30 epochs with batch_size=128.

After training for 30 epochs we get a decent Accuracy 85% on our test data.

2. model_final_res_2

After we have trained our based model with image size 128x128,we will transfer these weights to another inception_resnet_v2 model where we will train on the images of size 256x256.

This is a little trick I learned from Fast.ai course where Jeremy Howard trains on smaller images and then uses those weights to initialize same architecture to train larger images.

The reason why I think this techniques works is because when a smaller sized image is provided to our architecture,it tries to learn all the minute features which are constrained by the smaller image size.However once it learns these features under constraints,our model will eventually work well with larger sized images .This is just like **Data augmentation** which forces your model to learn better features,kind of like regularization.

So everything remains the same except we change our image size to 256X256 and train on the same conditions as above for 20 epochs.

Wallah ! we get an accuracy of 89.37% on our test data

model_final_res_2 will now become our base model.

INTRA-DOMAIN TRANSFER LEARNING

Transfer Learning involves the transfer of experience obtained by a machine learning model in one domain into another related domain . While document classification and object classification apparently seem like divergent domains, architectures trained on the 1000 class ImageNet dataset have proven to function as generalized feature extractors.

In this work, an inception_resnet_v2 model, trained on ImageNet, is used as initial weights of our holistic model thus constituting an initial level-1 (L1) transfer of weights. The L1 transfer, of course, originates from a different domain and is the regular inter-domain form of transfer learning. However, the holistic model trained on whole images of the RVL-CDIP dataset can be thought of as a generalized document feature extractor. The training sets for the region based models, although containing images of document regions and at a different scale, are still essentially images of documents. Thus, this concept is utilized by setting up another level (L2) of transfer learning in which the region based models are initialized with weights from the holistic model instead of the original inception_resnet_v2 model.

Using the concept of Intra-Domain Transfer learning we will now train **Region Specific** model by cropping our image in different ways.

Region Specific Models

1.Top-crop

We take the first 256 pixels from top to bottom and crop the remaining.

We do this by building a cropping function and passing that function to our generator.

```
def height_crop_generator(batches):
    while True:
        batch_X, label = next(batches)
        batch_Xcrops = np.zeros((batch_X.shape[0],256,256,3))
        for i in range(batch_X.shape[0]):
            batch_Xcrops[i] = batch_X[i][:256,:,:]
        yield (batch_Xcrops,label)
```

We now load the weights from the model_final_res_2 into a new inception_resenet_v2 model and train it for 5 epochs.

Using only the top region of the images we get 85.6%

2.Bottom-crop

We take the last 256 pixels from bottom to top and crop the remaining.

```
def bottom_crop_generator(batches):
    while True:
        batch_X, label = next(batches)
        batch_Xcrops = np.zeros((batch_X.shape[0],256,256,3))
        for i in range(batch_X.shape[0]):
            batch_Xcrops[i] = batch_X[i][-256:,:,:]
        yield (batch_Xcrops,label)
```

Using only the bottom region of the images we get 82.09%

3.Left-crop

We take the left most 256 pixels from left to right and crop the remaining.

```
def left_crop_generator(batches):
    while True:
        batch_X, label = next(batches)
        batch_Xcrops = np.zeros((batch_X.shape[0],256,256,3))
        for i in range(batch_X.shape[0]):
            batch_Xcrops[i] = batch_X[i][:,:256,:]
        yield (batch_Xcrops,label)
```

Using only the left region of the images we get 86.77%

4.Right-crop

We take the right most 256 pixels from right to left and crop the remaining.

```
def left_crop_generator(batches):
    while True:
        batch_X, label = next(batches)
        batch_Xcrops = np.zeros((batch_X.shape[0],256,256,3))
        for i in range(batch_X.shape[0]):
            batch_Xcrops[i] = batch_X[i][:,:256,:]
        yield (batch_Xcrops,label)
```

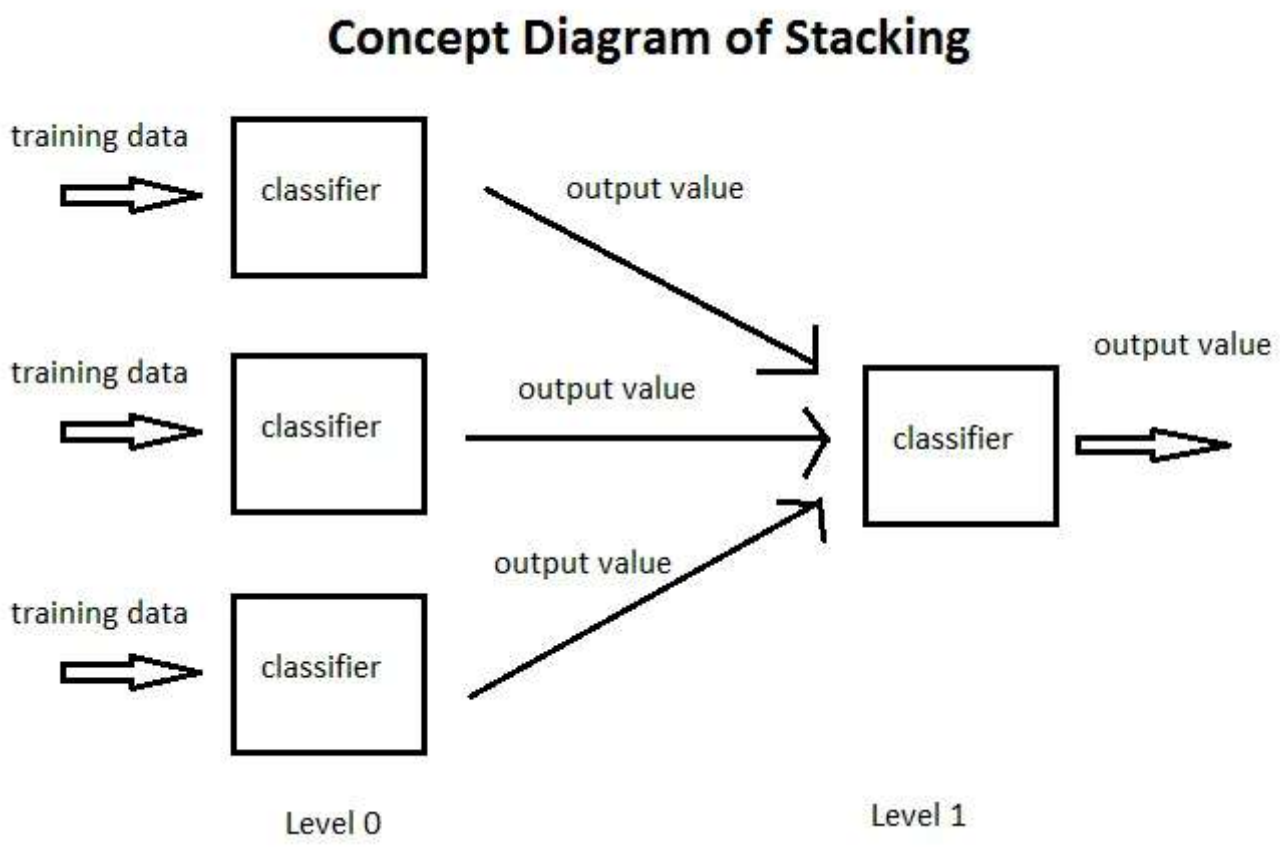
Using only the region of the images we get 84.36%

We have used Adam optimizer with 0.0001 Learning rate to train all the region specific models.

We have all the models trained by now.

Stacked Generalization

Lets use the power of Stacking to get SOTA results.



We take all our 5 models

1. Holistic model:model_final_res_2
2. Top crop model:mode_final_top
3. Bottom crop model:mode_final_bottom
4. Left crop model:mode_final_left
5. Right crop model:mode_final_right

Concatenate the softmax output of every model together for all the validation data and test data.

We will be training a simple 2-layered MLP Meta Classifier with the softmax output generated from the validation data and then we will test it on the test

data.

```
from keras.layers.normalization import BatchNormalization
from keras.layers import Dropout
from keras.initializers import RandomNormal
from keras.optimizers import RMSprop

model_relu = Sequential()
model_relu.add(Dense(128, activation='relu', input_shape=(80,)))
model_relu.add(Dropout(0.75))
model_relu.add(Dense(16, activation='softmax'))
```

```
WARNING:tensorflow:Large dropout rate: 0.75 (>0.5). In TensorFlow 2.x, dropout() uses dropout rate instead of keep_prob. Please ensure that this is intended.
```

```
print(model_relu.summary())
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_10 (Dense)             (None, 128)               10368
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_11 (Dense)             (None, 16)                2064
=================================================================
Total params: 12,432
Trainable params: 12,432
Non-trainable params: 0
_________________________________________________________________
None
```

```
from keras_radam import RAdam
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.3,
                              patience=2, min_lr=0.000001)
mcp_save = ModelCheckpoint('model_main.hdf5', save_best_only=True, monitor='val_accuracy', mode='max')
```

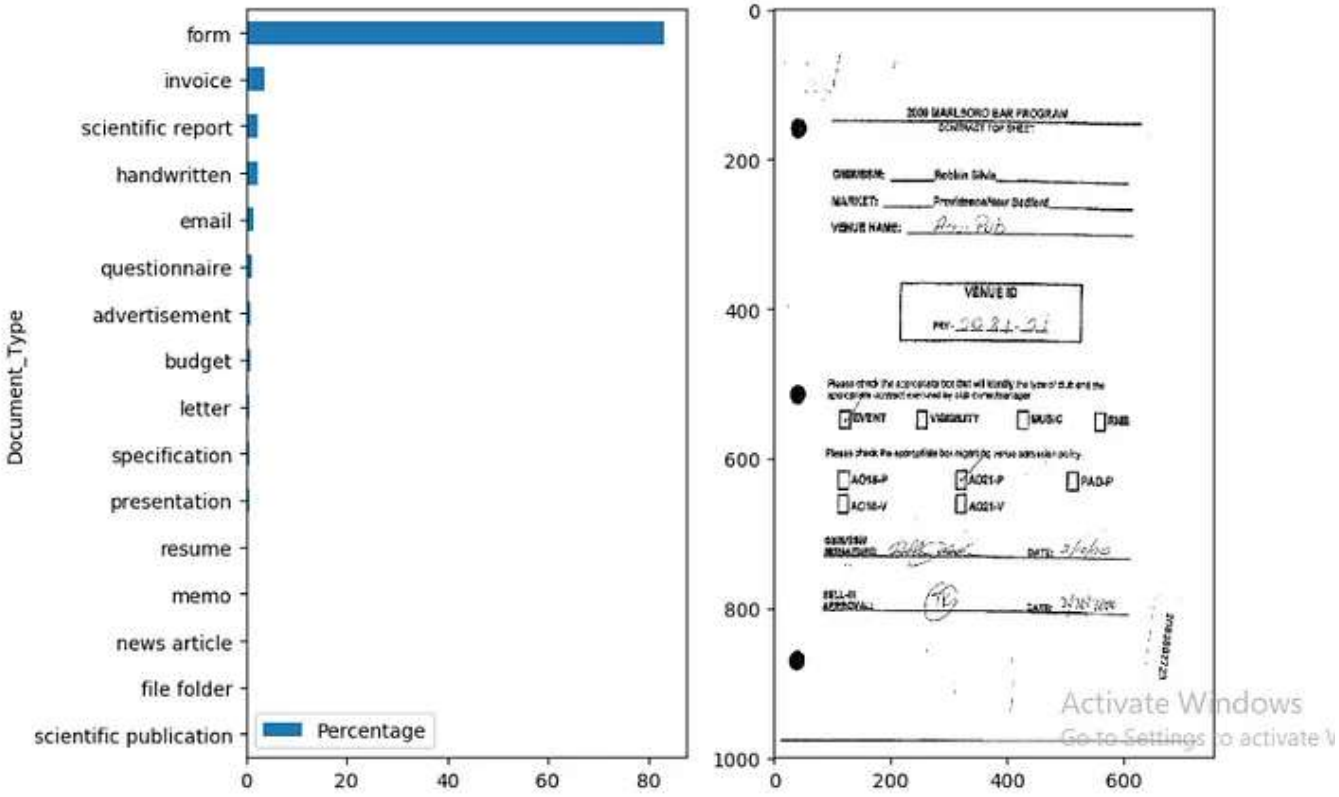
```
model_relu.compile(optimizer=keras.optimizers.Adam(lr=1e-3), loss='categorical_crossentropy', metrics=['accuracy'])

history = model_relu.fit(total_features,cv_labels, batch_size=256, epochs=10, verbose=1,validation_data=(total_features_test,test_labels),callbacks=[mcp_save])
```

After training for 20 epochs we get an accuracy of **90.5** % on both test and train data.

Pipeline

```
from matplotlib import cm
cmap = cm.get_cmap('tab20')
def height_crop(path,type):
    image=cv2.imread(path)
    image = cv2.resize(image, (512,256))
    if type=='bottom':
```

```
            image =image[-256:,:,:]

        else:

              image =image[:256,:,:]

         return image

    def width_crop(path,type):

       image=cv2.imread(path)
       image = cv2.resize(image, (256,512))
       if type=='right':

               image =image[:,:256,:]
       else:

               image =image[:,-256:,:]

       return image

    def full_image(path):

          image=cv2.imread(path)
          image = cv2.resize(image, (256,256))
          return image

    def preprocess(im):

         im = im/255
         im = np.expand_dims(im, axis=0)
         return im

    def predictions(image):

       top_pred=top.predict(image[0])
       bottom_pred=bottom.predict(image[1])
       left_pred=left.predict(image[2])
       right_pred=right.predict(image[3])
       holistic_pred=main.predict(image[4])
       total_features=np.hstack((top_pred,bottom_pred,
       left_pred,right_pred,holistic_pred))
       prediction=mlp_classifier.predict(total_features)

       return prediction

    def plot_bar_x(axes,prediction,doc_type):

           sort_index=np.argsort(prediction)[::-1]
           df=pd.DataFrame({'Document_Type':doc_type,
                            'Percentage':prediction})
           df=df.sort_values('Percentage',ascending=True)
           labels=df['Document_Type']
           df.plot(kind='barh',y='Percentage',x='Document_Type',
    ax=axes,colormap=cmap)
```



Our output for a random image is spot on !!

Conclusion

We achieved an accuracy of 90.5% with 1/3rd data using Intra-domain transfer learning followed by Stacked Generalization.

Using more data could produce better results and I am working on that.

Ill keep you posted once we beat the 92.4 benchmark soon !!

You can get the entire code at : https://github.com/sambalshikhar/Document-Image-Classification-with-Intra-Domain-Transfer-Learning-and-Stacked-Generalization-of-Deep

Caution : The notebook shows the final results to be 93% which is an error of not resetting the Generators.After fixing we found the accuracy to be 90.5%.

Credits:

Document Image Classification with Intra-Domain Transfer Learning and Stacked Generalization of...

In this work, a region-based Deep Convolutional Neural Network framework is proposed for document structure learning...

arxiv.org

https://ai.googleblog.com/2016/08/improving-inception-and-image.html

Machine Learning | Deep Learning | Keras | TensorFlow | Data Science

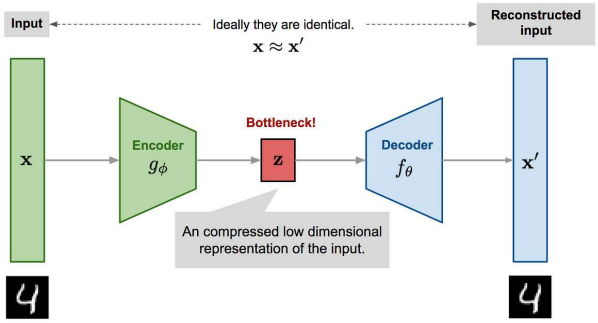
Written by sambal shikhar

Follow

8 Followers · Writer for Analytics Vidhya

Back propagating my parameters.

More from sambal shikhar and Analytics Vidhya



sambal shikhar

Auto-Encoders And The Battle Of Generations

The generative models have taken over the world by storm ! Be it the controversial Deep...

11 min read · Oct 24, 2019

11

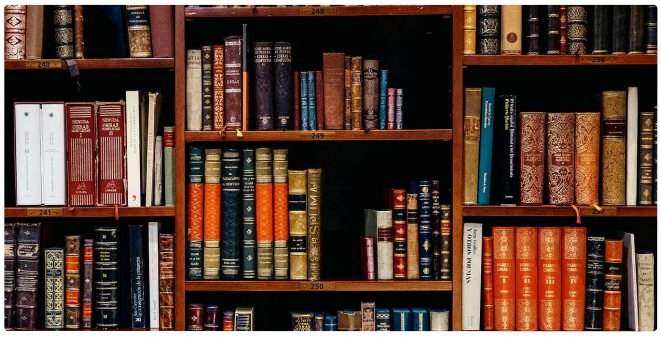
Sajal Rastogi in Analytics Vidhya

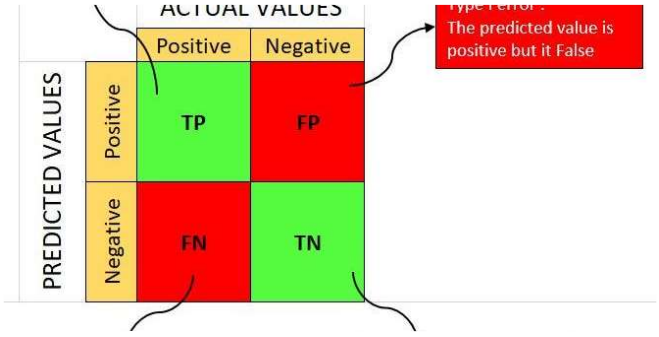
Wifi -Hacking using PyWifi

.

4 min read · Feb 6, 2021

280 2





Kia Eisinga in Analytics Vidhya

How to create a Python library

Ever wanted to create a Python library, albeit for your team at work or for some open...

7 min read · Jan 26, 2020

2K 24

Anuganti Suresh in Analytics Vidhya

What is a confusion matrix?

Everything you Should Know about Confusion Matrix for Machine Learning

8 min read · Nov 17, 2020

627 5

See all from sambal shikhar

See all from Analytics Vidhya

Recommended from Medium

Khang Pham

Text Classification with BERT

In this tutorial, we will use BERT to develop your own text classification model.

8 min read · May 9

45 1

Rajat Roy in MLearning.ai

Document Intelligence Series— Part-1: Table Detection with YOLO

How to perform table detection on Bordered and Borderless table document with YOLOv8

3 min read · Aug 13

66

Lists

Predictive Modeling w/ Python

20 stories · 512 saves

Practical Guides to Machine Learning

10 stories · 582 saves

Natural Language Processing

735 stories · 326 saves

New_Reading_List

174 stories · 152 saves

Eugenia Anello in Towards Data Science

Top 5 Python Libraries for Extracting Text from Images

Understand and master OCR tools for text localization and recognition

✨ · 7 min read · Jul 25

177 1

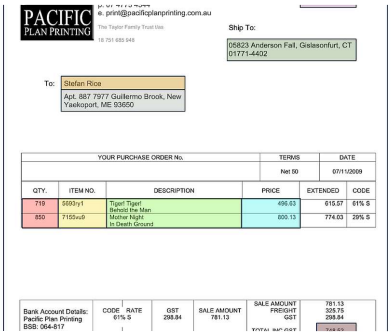
Aravinda 加阳

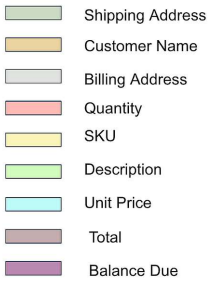
How to split image dataset into train, validation and test set?

Splitting image data into train, validation, and test sets is a crucial step in machine learning...

1 min read · Apr 23

9 1





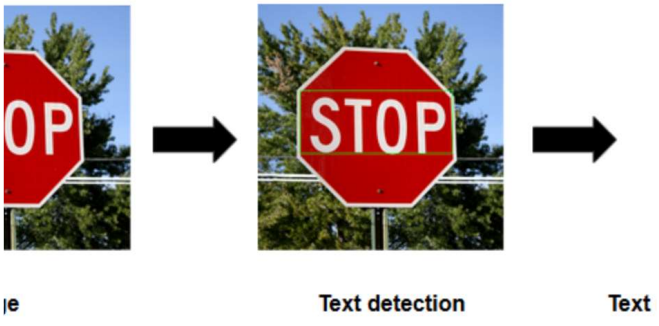
Ravi Satvik

Unleashing the Power of LayoutLM: Extracting Entities from Structure...

A Comprehensive Guide to Harnessing the Power of LayoutLM for Structured Documen...

6 min read · May 24

8



Drumil Shah in Searce

Exploring Text and Table Extraction Packages in Python

Introduction

9 min read · Jul 7

16

See more recommendations